

Introduction to VHDL

Coding for Synthesis

Jim Frenzel

VHDL

- ❑ VHSIC Hardware Description Language
- ❑ NOT a programming language!
- ❑ Hierarchical Modeling
- ❑ Hardware Abstraction
- ❑ Synchronous and Asynchronous Logic
- ❑ Combinational and Sequential Logic

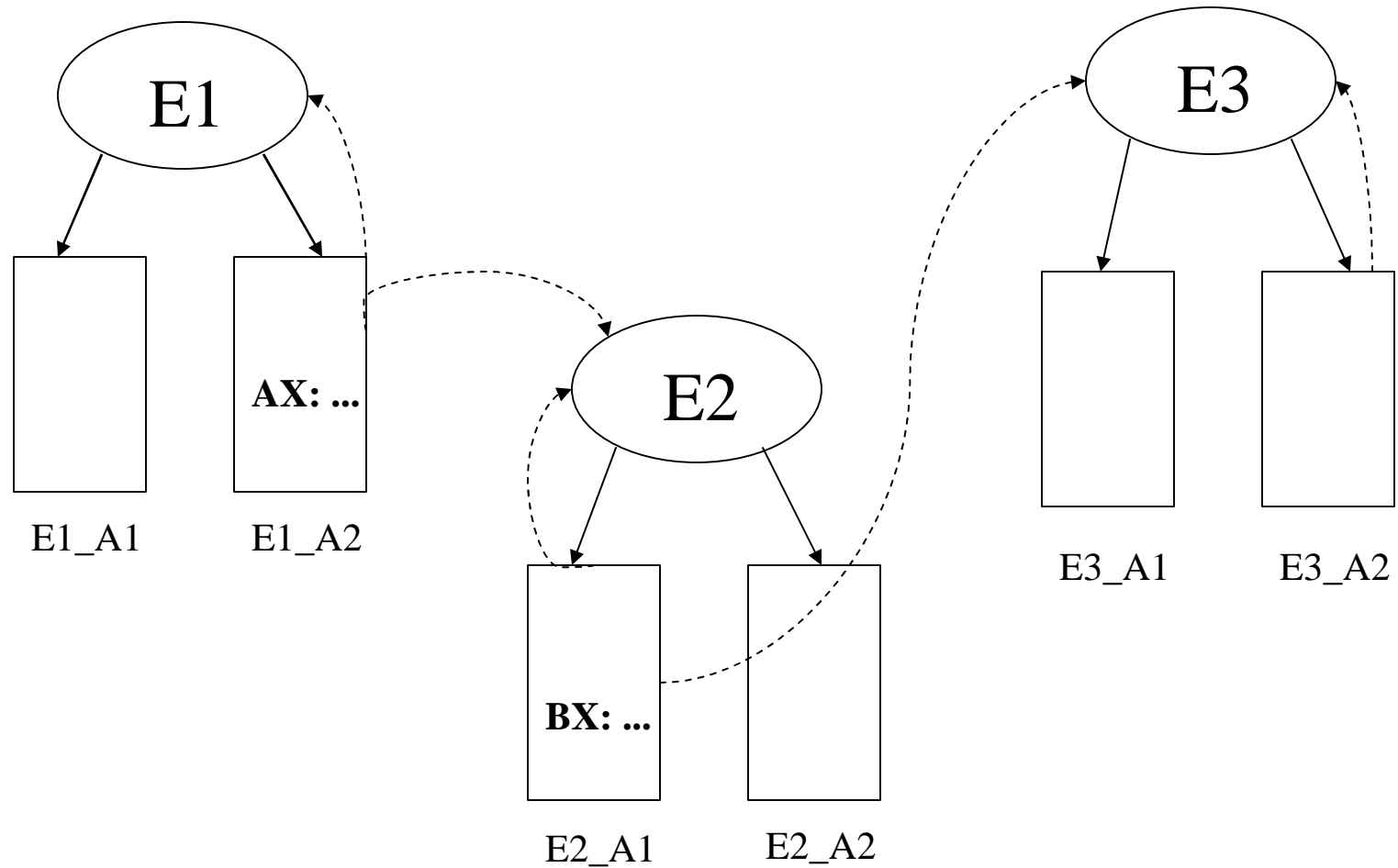
Hierarchy

- ❑ A design consists of components
- ❑ Each component has an entity and one or more architectures
- ❑ The entity describes the I/O interface
- ❑ The architecture models the function of the component
- ❑ When a component is used in a design (instantiated), an entity-architecture pair is specified.

Design Flow

- ❑ Produce the VHDL Model
- ❑ Compile the VHDL
- ❑ Simulate the compiled VHDL
- ❑ Synthesize/Optimize to a technology
- ❑ Simulate again

Configuration



Entity Port Types

- ❑ In: value read within entity model
- ❑ Out: value updated within entity model; cannot be read.
- ❑ Inout: bidirectional port; can be read and updated.
- ❑ Buffer: can be read and updated within the entity model.
Can only have one source.

Modeling Styles

- ❑ Structural: Used to describe the interconnection of primitive and user-defined components.
- ❑ Dataflow: Provides logical and arithmetic operators.
- ❑ Behavioral: Allows modeling using if-then, case statements, etc.

Architecture

- ❑ Interconnected components (structure)

- ❑ Concurrent Statements (dataflow)

Executed in “parallel”. Order independent.

- ❑ Sequential Statements (behavior)

Executed sequentially inside a process. Order dependent.

Identifiers

- ❑ (A-Z), (a-z), (0-9), underscore.
- ❑ First character is a letter, last not an underscore.
- ❑ Two consecutive hyphens starts a comment.
- ❑ Case insensitive. (except for type values, e.g., 'Z')

Data Objects

- ❑ Constants
- ❑ Variables
- ❑ Signals: Maintain a history. Permits modeling of delays, detection of edges.

Types

- ❑ Enumerated Types
- ❑ Single bit and bit-vector signals
- ❑ Symbolic States

Operators

- ❑ Logical: and, or, nand, nor, xor, not, xnor (VHDL-93)
- ❑ Relational: =, /=, <, <=, >, >=
- ❑ Adding: +, -, &
- ❑ Multiplying: *, /, mod, rem
- ❑ Misc: abs, **

Architecture Revisited

```
architecture [name] of [entity name] is
    [declarations]
begin
    concurrent statements;
    (signal assignments, process statements
     component instatiations)
end [name];
```

Process Statement

```
[process label:] process [(sensitivity list)]  
    [process declarations]  
begin  
    sequential statements;  
    (signal assignments, if statement, case statement)  
end process [process label];
```

If Statement

```
if (boolean expression) then
    sequential statements;
elsif (boolean expression)
    sequential statements;
else
    sequential statements;
end if;
```

Note the weird spelling of “elsif”!

Case Statement

```
case (expression) is
  when (choices) => sequential statements;
  [...]
  when others => sequential statements;
end case;
```

Tips from the Master

- ❑ Think hardware. If you don't know what it will synthesize to, don't write it.
- ❑ Specify signal values under all occurrences (end with “else” or “when others”). Specify default at the start of a process.
- ❑ Separate out next state logic and output logic (or state flip-flops and combinational logic).
- ❑ Partition, partition, partition!

```

entity FULL_ADDER is
    port (A, B, CIN: in BIT; SUM, COUT: out BIT);
end FULL_ADDER;
architecture FA_MIXED of FULL_ADDER is
    component XOR2
        port (P1, P2: in BIT; PZ: out BIT);
    end component;
    signal S1: BIT;
begin
    X1: XOR2 port map (A, B, S1);
    process (A, B, CIN)
        variable T1, T2, T3: BIT;
    begin
        T1 := A and B;
        T2 := B and CIN;
        T3 := A and CIN;
        COUT <= T1 or T2 or T3;
    end process;
    SUM <= S1 xor CIN;
end FA_MIXED;

```

- structure

- behavior

- dataflow

Figure 2.7 A 1-bit full-adder.

